Reliable Computation in Computing Systems Designed from Unreliable Components*

By MICHAEL G. TAYLOR

(Manuscript received April 10, 1968)

This is the second of two papers which present information-theory-type results pertaining to the reliability of computing systems designed from unreliable components. Two models for component malfunctions are considered. The first is based on the assumption that malfunctions of a particular component are statistically independent from one use to another. The second is based on the assumption that components fail permanently but that the components which have failed are periodically replaced with good ones. In both cases, malfunctions in different components are assumed to be independent. Just as a channel capacity is defined for communication channels, a computing capacity is defined for computing systems. For both component failure models, it is shown that there are computing systems, designed entirely from unreliable components of the assumed type, which have nonzero computing capacities.

I. INTRODUCTION

The objective of this paper is to show that it is possible for a computer, designed entirely from unreliable components, to perform operations reliably on information stored in stable memories. The concept of a stable memory is introduced in the paper preceding this, where it is shown that it is possible to store information reliably in memories constructed entirely from unreliable components.[1] Two different models for component malfunctions are considered. The first is based on the assumption that component malfunctions are statistically independent from one use of a particular component to another use. The second is based on the assumption that components

* This work, which is based on part of a doctoral thesis submitted to the Department of Electrical Engineering, M.I.T., September 1966, was supported by the National Aeronautics and Space Administration (Grant NsG-334).

fail permanently but that the components which have failed are periodically replaced with good ones. In both cases component malfunctions are assumed to be statistically independent from one component to another. For both component malfunction models it is shown that there are types of memories, called "stable memories," that have a nonzero information storage capacity; that is, for certain fixed values of the memory's redundancy, the probability of a memory failure can be made arbitrarily small. A particular type of stable memory is considered in some detail.

Since the operation of the computers to be described in this article is closely related to the operation of these stable memories, let us look briefly at these memories. They consist of several registers and a correcting network as shown in Fig. 7 of Ref. 1. The registers store information which is coded according to a low-density parity-check code[2] and the correcting network periodically monitors the contents of the registers, corrects errors and reinserts the corrected words into the registers. The correcting network is very similar to a low-density parity-check decoder. Such a decoder, if constructed from reliable components, decreases the probability of error for digits stored in the registers with each successive correcting cycle (iteration) provided that the initial probability of error is not too large.

In order to understand the operation of a low-density parity-check corrector constructed from unreliable components, let us suppose that the initial probability or error for stored digits is somewhat higher than the probability of malfunction for any component in the corrector. For the first few interations the corrector decreases the probability of error per digit almost as much as the reliable decoder does. However, eventually this probability becomes comparable to the probability that a new error is made by the correcting network itself. Thus an equilibrium probability of error per digit is established such that the probability of error per digit before and after each correcting cycle is the same. Figure 1 is typical graph of this probability.

Although the probability of error per digit approaches an asymptotic value, it is still possible for a memory failure to occur. A memory failure occurs whenever the configuration of errors within the registers of the machine is such that a noiseless decoder would be unable to correct all the errors. The probability of a memory failure during the time interval $0 \leqq t \leqq \mathcal{L}\tau$ is calculated in Ref. 1 and it is shown to have the same form for both component malfunction models considered. For any k and $\mathcal{L}$, the probability of a memory failure in M_k, a memory of the type under

consideration which can store k bits of information, is given by

$$\Pr[\text{failure of } M_k \text{ during } 0 \leqq t \leqq \mathfrak{L}\tau] < (\mathfrak{L} + 1) \cdot C' \cdot k^{-\beta'}$$

where C' and β' are constants for any particular type of memory, or to be more mathematically precise, for any particular sequence of memories, $\{M_i\}$, where the members of the sequence are ordered according to their information storage capabilities. It also is shown that it is possible to make $\beta' > 0$; in fact, a numerical example is presented where $\beta' = 5.55$. Therefore, for this example, it is possible to make the probability of a memory failure arbitrarily small by choosing k, the number of bits stored, sufficiently large. Furthermore, it is shown that increasing k does not increase the redundancy of the memories being considered, thus completing the proof of stability for these memories.

Our objective is to show that it is possible to design a reliable computer which performs arithmetic operations on operands stored in stable memories and which presents the result in a form that can itself be stored in a stable memory. The latter condition assures that successive arithmetic operations can be performed by computers of this type. Just as stability is defined to be a property of sequences of memories, reliability will be defined to be a property of sequences of computing systems.

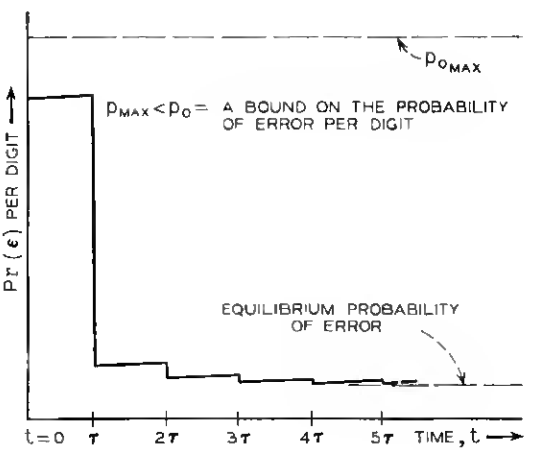

Fig. 1 — The probability of error per digit.

II. DEFINITION OF RELIABILITY

The computing systems we are considering consist of a number of stable memories for information storage and some logic circuits which perform the arithmetic operations. The operations are performed on operands which have been stored in stable memories and the results of the operations are stored in stable memories. Before it is possible to define "reliability," it is necessary to establish a criterion for determining whether a particular result is correct. We consider a result (coded) to be correct if a noiseless low-density parity-check decoder could correct all the errors; that is, if it could obtain the desired uncoded result. Thus, the class of correct results is precisely the decoding equivalence class which contains the code word whose information digits correspond to the desired uncoded result.

Now consider a sequence of computing systems $\{S_i\}$ where, for any k, the computing system S_k contains memories having an information storage capability of k binary digits. We require that all the computing systems in $\{S_i\}$ be able to perform the same set of operations. The sequence $\{S_i\}$ is called *reliable* if it satisfies the following conditions:

(*i*) For all k, the redundancy of S_k must be less than α where α is a constant independent of k. The redundancy of a computing system is defined as the ratio of the complexity of the system to the amount of computation performed by the system.*

(*ii*) For any $\mathfrak{I} > 0$ and $\delta > 0$, there must be a member of $\{S_i\}$ for which the probability that the result of any sequence of $\mathfrak{I}$ operations (within the allowed set) will be in error is less than δ.

The reciprocal of the minimum redundancy for which a particular sequence of computing systems is reliable is called the *computing capacity* for these systems.

III. OPERATION OF VECTOR ADDITION MODULO-2

The operation of vector (bit-by-bit) addition modulo-2 is considered first because low-density parity-check codes have the property that when this operation is performed on two code words, the result is always another code word. This is the only nontrival operation that can be performed without using some elaborate procedure for generating the check digits required to form the coded result. For

* Ref. 1 gives precise definitions of "complexity" and "amount of computation."

this reason, the operation of modulo-2 addition is particularly easy to perform.

The computing system to he considered consists of stahle memories which store the operands and the results, and logic circuits which perform the arithmetic operation. It is assumed that all components within the computing system have a nonzero probability of malfunction. Let the stable memories containing the operands for one particular operation be denoted by M_k' and M_k'' . At $t = T$, the operation of vector addition modulo-2 is performed on the contents of these memories and the result is stored in another stahle memory. It is assumed that all the memories within a particular computing system are physically identical. This means that all memories must have the same set of possihle states; furthermore, since the states of a stahle memory are divided into classes of equivalent states, the classes of states must he the samc for all of these stable memories. Let us suppose that at $t = T$, the state of M_k' helongs to $C(I_{ki_1})$, the elass of states containing the code word I_{ki_1} , and that the state of M_k'' helongs to $C(I_{ki_2})$. If the operation is performed correctly, the state of the stable memory containing the result will helong to $C(I_{ki_1} \oplus I_{ki_2})$.

To show that computing systems of this type are reliahle, we must show that a sequence of 3 operations can he performed with an overall prohahility of error that can be made arbitrarily small by choosing k sufficiently large while keeping the redundancy fixed. To compute the redundancy we evaluate the ratio of the complexity of the system to the amount of computation performed hy the system. Each modulo-2 vector addition performed hy the system involves three stable memories, each with an information storage capability of k, and a numher of modulo-2 adders equal to the number of information storage components in one stahle memory.

In Ref. 1 it was shown that the complexity of a stahle memory is proportional to k; hence the complexity of these three memories and the associated modulo-2 adders must also be proportional to k. The amount of computation, that is, the number of two-input hinary operations that an equivalent irredundant computer would perform, equals k since this irredundant computer would perform k additions (modulo-2) on its two k-bit operands. Sinee both the complexity of the systems heing considered and the amount of computation that they perform are proportional to k, their ratio, the redundancy, is independent of k as required.

Let us start hy considering just one operation performed hy the computing system. We assume that hefore the operation was performed

hoth M_k' and M_k'' performed at least one correcting cycle on the stored operands. Thus, according to Fig. 1, the probability of error for digits stored in each of these memories is upper bounded hy p_1 which, in general, is very small compared with the maximum allowable prohability of error per digit. After the operation has been performed, the probability of error (ϵ) for digits in the memory storing the result is upper hounded hy

$$\Pr[\text{digit} = \epsilon] < 2p_1 + p_a + 2p_r$$

since any one of the following events could lead to an error in one particular digit:

(*i*) Tbe corresponding digit was in error in M_k'(prohahility $\leqq p_1$).

(*ii*) The corresponding digit was in error in M_k''(prohahility $\leqq p_1$).

(*iii*) The adder that performed the operation on these digits made an error (probahility denoted hy p_a).

(*iv*) An error occurred in this particular digit position in the result memory hetween the time the result was stored and the end of the first correcting cycle performed on the result (probahility $\leqq 2p_r$).

Provided that the resulting probability of error per digit is less than the maximum allowahle value, successive iterations of the result memory decrease this probahility as shown in Fig. 1. After the result memory has performed one iteration, the contents of this memory can be used as an operand in a second operation. Successive operations can he performed provided that at least one correcting cycle is performed on each intermediate result.

An error is made on one of these operations if the state of the result memory does not helong to the desired class of states, or equivalently, if a memory failure occurs within the result memory immediately following the operation. The method for computing the prohahility of such an error is almost identical to that used in Ref. 1 to compute the prohahility of a memory failure. The result, which is obtained in the Appendix, is:

$$\Pr[\text{any } \oplus \text{ operation is performed incorrectly} \mid \text{all previous } \oplus \text{ operations were performed correctly}] < \frac{C}{(1 - J/K)^2} \cdot k^{-\beta+3}$$

where C and β are functions of the parameters of the code (J and K) and p_0, the bound on the probability of error per digit. For any par-

ticular sequences of computing systems, J, K, and p_0, and hence C and β, are all constants. This result applies for either of the component malfunction models discussed in the introduction.

Finally we must bound the probability that an error occurs on any one of a sequence of 3 operations. It is assumed that neither a memory failure nor a propagation failure occurred in any stable memory between the time when an operand was originally stored and the time when the first operation was performed on that operand. We need not concern ourselves about the concept of a propagation failure except to notice that the bounds on the probability of a memory failure were actually derived by bounding the probability of either a memory failure or a propagation failure. Hence, imposing the condition of no propagation failure does not make the requirements any different from those which have already been used. The probability of an error during 3 operations is upper bounded by the sum of the probabilities that the initial error occurs on any one of these operations; that is,

$$\Pr[\text{error during a sequence of } 3 \oplus \text{ operations} \mid \text{no memory failure or propagation failure in memories containing the original operands}] < 3 \cdot \frac{C}{(1 - J/K)^2} \cdot k^{-\beta+3}.$$

If $J = 14$, $K = 15$, and $p_0 = 10^{-8}$ then $\beta = 7.55$; therefore, for this sequence of computing systems, the probability of an error in 3 vector modulo-2 additions can be made arbitrarily small by choosing k sufficiently large, thus providing that this sequence of computing systems is reliable.

IV. GENERAL VECTOR OPERATIONS

Consider a sequence of computing systems, $\{S_i\}$, in which each system is capable of performing many different operations. The inputs to each of these systems are stored in stable memories, as before; however, the inputs must specify not only the operands but also the desired operations. The digits stored in each memory are coded according to a low-density parity-check code, but now it is assumed that the code is in systematic form; that is, the information digits appear first in each code word. When an operation is performed on two code words, the desired result is the code word whose information digits are computed by performing the desired operation on the information digits of the two operands.

The allowed operations for the computing systems under consideration consist of any vector (hit-hy-hit) operation on the information digits of the operands. That is, the operation pcrformed on a particular pair of information digits in the operands can he any one of the 16 Boolean functions of two variahles. Each of these operations is assigned a four-hit operation code. Since different operations can he performed on different pairs of information digits in the operands, the computing system S_k , which contains memories having a storage capahility of k hits, is ahle to perform $(16)^k$ different operations on any pair of operands. The desired operation is specified hy means of four code words. For all $0 < i \leqq k$, the set of four digits in the ith digit position in each of these four code words gives the operation code for the operation to he performcd on the ith information digit in each operand. For each pair of information digits in the operands, the computing system first selects the operation specified hy the operation code and then performs this operation on the appropriate pair of digits in the operands. In this way all the information digits in the result are computed.

Since the dcsired result is a code word, it is necessary for the computing system to generate the appropriate check digits to go with the desired information digits of the result. The operation of vector addition modulo-2 is particularly easy to perform on the information digits of two code words because the appropriate check digits can he generated by performing the operation of vector addition modulo-2 on the check digits of the two operands. In general, it is more difficult to generate the appropriate check digits.

We consider first a method hy which a noiseless system could generate the check digits, then show that hy making a rather simple modification to this method, it is suitahle for use in a noisy system. Finally we hound the prohability that an error occurs in a sequence of 3 operations performed using this modified method. To simplify the terminology throughout this and suhsequent discussions, we contrast noiseless and noisy systems. A noiseless system is one in which there are no component malfunctions whereas, for these discussions, a noisy system is any one in which the components have nonzero prohahilities of malfunction. All results presented apply for either of the component malfunction models descrihed in the introduction.

Let us consider how a noiseless system consisting of memories, operation selectors, and computing devices might compute the desired result. Since there are 16 operations that could hé performed on any pair of information digits in the operands, there must he 16 types of computing devices. One device of each type and one operation selector

is associated with each pair of information digits in the operands. Each operation selector decodes the operation code and selects the appropriate computing device. This device then performs the desired computation on one pair of information digits. In this way all the information digits in the desired result are computed. These resulting information digits are represented by the row vector $\mathbf{v}$.

The second step in computing the desired result is to compute the check digits. Let us assume that, within the computing system, k noiseless memories are used to store a set of k basis vectors for the code. This set of k basis vectors is represented by the generator matrix, $\mathbf{G}$. It is assumed that the basis vectors have been chosen in such a way that $\mathbf{G}$ is in reduced-echelon form. (That is, $\mathbf{G} = [\mathbf{I}_k\mathbf{P}]$ where $\mathbf{I}_k$ is a $k \times k$ identity matrix and $\mathbf{P}$ is a $k \times (N - k)$ matrix. The notation $\mathbf{D} = [\mathbf{AB}]$ means that the matrix $\mathbf{D}$ can be partitioned into two submatrices, $\mathbf{A}$ and $\mathbf{B}$.)[3] The desired result is obtained by performing the operation $\mathbf{v} \odot \mathbf{G}$ where $\odot$ represents the operation of matrix multiplication in which all additions are performed modulo-2.

This matrix multiplication operation is performed in two steps. The first step consists of performing the bit-by-bit AND of the digits represented $\mathbf{v}^T$ with the digits represented by each column of $\mathbf{G}$ (see Table I). The resulting row vectors, represented by the matrix $\mathbf{G}'$, are stored in another set of k noiseless memories. A row of $\mathbf{G}'$ is all zeros if the corresponding digit in $\mathbf{v}^T$ is 0 but it is identical to the same row of $\mathbf{G}$ if the corresponding digit in $\mathbf{v}^T$ is 1. Therefore, the nonzero rows of $\mathbf{G}'$ are the generators of the desired result. The final step in performing the matrix multiplication operation consists of adding modulo-2 the rows of $\mathbf{G}'$.

Let us see if a noisy computing system, containing stable memories rather than noiseless memories, can obtain the desired result in the

TABLE I—AN EXAMPLE OF THE OPERATION $\mathbf{v} \odot \mathbf{G}$.

$$\mathbf{v} = [1 \quad 0 \quad 1]$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$\mathbf{v}^T$ ANDed with columns of $\mathbf{G}$

$$= \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\triangleq \mathbf{G}'$$

Vector sum modulo-2 of the rows of $\mathbf{G}'$

$$= [1 \quad 0 \quad 1 \quad 1 \quad 0]$$

$$\triangleq \mathbf{v} \odot \mathbf{G}$$

same way. The information digits of the result are computed by the method described above. This time, however, there is a nonzero probability that any particular information digit is in error. There is now a problem when we try to use these digits to find the generators of the desired result. If there is any error in these information digits, the generators of the result will be specified incorrectly, in which case the resulting state of the stable result memory will almost certainly be outside the desired class of states. Since there is no way to eliminate the errors in the information digits, the computing method as described is not suitable for use in a noisy computing system.

The problem with the computing method that has just been described is that there is only one copy of the information digits of the result and there is no way to guarantee that this copy is error free. Suppose that we had many copies of these information digits such that the errors were statistically independent from one copy to another. We shall show later that it is actually possible to obtain copies with these properties. These copies of the desired information digits can be represented by means of a matrix $\mathbf{V}$ in which each column represents one of these copies. Each row of $\mathbf{V}$ is almost all zeros or almost all ones and the errors across any row of $\mathbf{V}$ are statistically independent. Each row of $\mathbf{V}$ that is almost all ones indicates that the corresponding row of the generator matrix is one of the generators of the desired result. Suppose that the generators of the code are stored in stable memories, the generator set. Since each stable memory containing k bits of information actually contains $J \cdot N$ binary digits (N is the block length of the code and J is a parameter of the code described previously), the contents of these stable memories in the generator set can be represented by a $k \times JN$ matrix $\mathcal{G}$. If we require that $\mathbf{V}$ be a $k \times JN$ matrix too (that is, that we have JN copies of the information digits), an operation can be performed on the memories equivalent to ANDing corresponding entries in the matrices $\mathcal{G}$ and $\mathbf{V}$, the result being represented by a $k \times JN$ matrix $\mathcal{G}'$. This operation leaves the desired generators essentially unchanged but it replaces the undesired generators with vectors which are equivalent to the zero vector.

In this case an error in one copy of the information digits causes at most one error in the digits represented by $\mathcal{G}'$. This is much better than the result of the previous method where one error in the information digits causes errors in an entire row of $\mathbf{G}'$. If the probability of error per digit in $\mathcal{G}'$ is not too high, it would be hoped that each stable memory corresponding to a row of $\mathcal{G}'$ would be able to reduce this probability of error by performing one correcting cycle. The operation of vector addition modulo-2 could then be performed to obtain the desired result. Since

this method looks promising, it will be considered in more detail. The first step is to make sure that it is possible to obtain copies of the information digits with the desired properties. Then a sequence of computing systems which perform operations by this method is analyzed to determine if the sequence can be made reliable.

When the computing system is ready to operate on a particular operand it copies the operand JN times and each copy is stored in a stable memory. A set of JN stable memories corresponding to one operand is called an operand set (see Fig. 2). An operand matrix, $\mathbf{A}_i$ $i = 1, 2$, is defined in the following way. Each column of $\mathbf{A}_i$ is associated with one stable memory in the corresponding operand set. The digits in a particular column of $\mathbf{A}_i$ equal the digits in the first noisy register within the stable memory corresponding to that column of $\mathbf{A}_i$. Recall that each register within a stable memory contains a noisy approximation to the same code word. Thus the digits in each row of an operand matrix are almost all ones or almost all zeros. The digits different from the dominant one in each row correspond to errors in different stable memories within the operand set. Let each operand memory perform m (m = the number of independent iterations[2]) iterations on the digits contained in it. If no memory failure or propagation failure occurs in any stable memory within an operand set, the errors in the digits stored

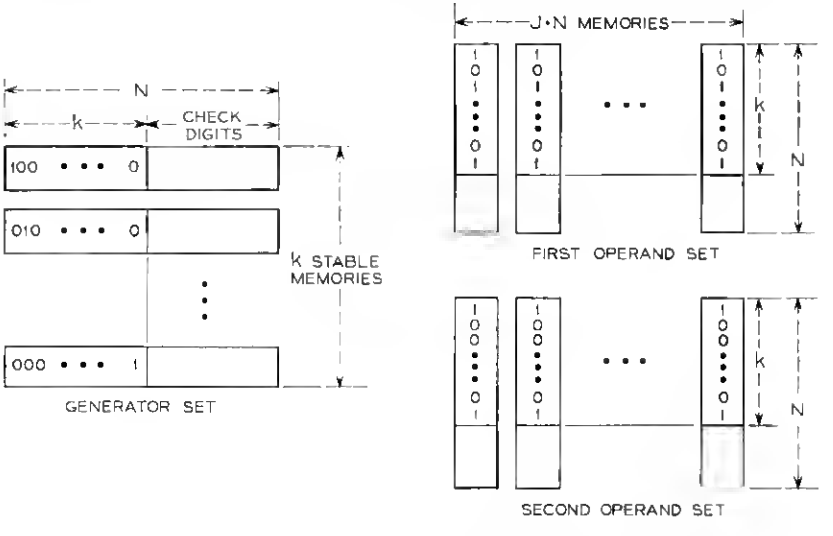

Fig. 2 — The generator set and typical operand sets. The digits are typical of the digits in each of the J noisy registers within the stable memory.

within a stable memory depend only on the component errors that occurred in that memory during the last m iterations,[1] and these component errors are assumed to be statistically independent from one stable memory to another. Since the contents of these stable memories are represented by the columns of the operand matrix $\mathbf{A}_i$, the errors in one column of this matrix are statistically independent of the errors in any other column. Therefore, after m iterations, the errors across any *row* of the operand matrix are statistically independent.

The four stable memories which contain the code words that specify the operation code are called the "operation code memories." The contents of each of these memories are copied JN times to form an operation code set (see Fig. 3). Each memory in each operation code set performs m iterations just as the memories in each operand set do. An operation code matrix $\mathbf{O}_i$, $0 < i \leqq 4$, is defined for each operation code set; the definition is analogous to the definition of each operand matrix as stated previously.

Let us consider a particular digit position in each of the operation code matrices and each of the operand matrices. The four digits in this position in the four operation code matrices form the operation code for the operation to be performed on the two digits in this position in the two operand matrices. After each memory in each set of memories has performed m iterations, the operations specified by the operation code matrices are performed. The digits which are the results of these opera-

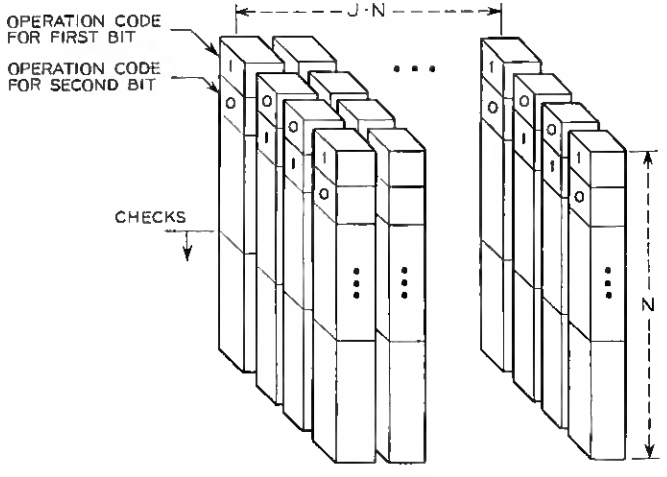

Fig. 3 — The operation code sets.

tions can be represented by $k \times JN$ matrix, each column of this matrix being an approximation to the information digits of the desired result.

Since the errors in each row of each operand matrix and each operation code matrix are statistically independent, and since the errors made by the components which perform the operations are statistically independent, the errors along any row of this resulting matrix must be statistically independent. Therefore, this is precisely the matrix $\mathbf{V}$ described previously. The digits represented by this matrix are ANDed with the digits represented by $\mathfrak{G}$ to form the matrix $\mathfrak{G}'$. The operation of vector addition modulo-2 is then performed on the digits in the registers represented by the rows of the matrix $\mathfrak{G}'$ to obtain the desired result (see Fig. 4).

Finally we must compute the probability that all of the operations are performed in such a way that the state of the result memory belongs to the desired class of states. The first step required that m iterations be performed by each memory in the operand sets and the operation code sets. There are a total of $6 \cdot J \cdot N$ stable memories in these sets. The memories represented by the generator matrix $\mathfrak{G}$ must also perform m iterations. There are k memories in this generator set. It is assumed that neither a memory failure nor a propagation failure occurred in any stable memory before the m iterations were started. We wish to bound the probability that a memory failure or a propagation failure occurs in any memory during these m iterations.

In Ref. 1 the probability that either a memory failure or a propagation failure occurs in any one memory on any particular iteration, given that no memory failure or propagation failure occurred previously, was upper bounded. This bound equals

$$\Pr[\text{memory failure or propagation failure} \mid \text{no previous memory failure or propagation failure}] < \frac{C}{1 - J/K} k^{-\beta+2}$$

where C and β depend on the parameters of the code and the probabilities of component errors but not on k. This bound has the same form for either of the two component malfunction models discussed in the introduction.

Since

$$N \leqq \frac{k}{1 - J/K} \quad \text{and} \quad m < \log \left[\frac{k}{1 - J/K} \right] \quad \text{for} \quad k > 2,$$

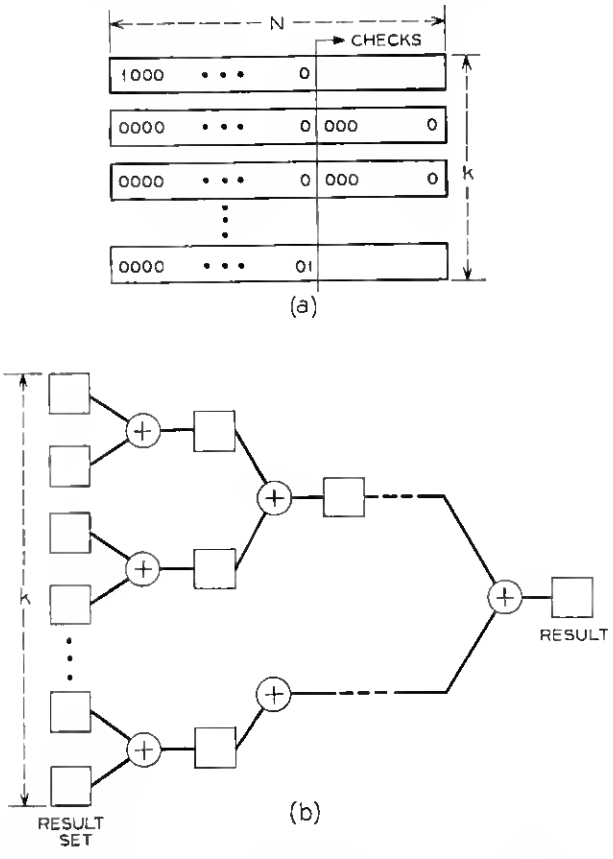

Fig. 4 — (a) The result set corresponding to matrix $\mathcal{G}$ (illustrates the vector AND of the two operands). (b) The addition operation.

the probability that a memory failure or a propagation failure occurs during the m iterations is bounded by

Pr[memory failure or propagation failure during m iterations]

$$< \left\{6\left[\frac{Jk}{1 - J/K}\right] + k\right\} \cdot \left\{\log\left[\frac{k}{1 - J/K}\right]\right\} \cdot \frac{C}{1 - J/K} \cdot k^{-\beta+2}$$

for $k > 2$.

The operation is performed following these m iterations. The result of this operation should be the generators of the desired result, each generator being stored in a stable memory. These memories are called the result set and are represented by the matrix $\mathcal{G}'$. Let us suppose that

no memory failure or propagation failure occurred during the m iterations. We now bound the probability that a digit within a memory in the result set is in error just before the time when the memory has completed the first correcting cycle on the newly inserted digits. If one of these digits is in error, at least one of the following events must have occurred:

(*i*) The corresponding digit was in error in either one of the operand sets. The probability of this event is bounded by $2p_1$ (see Fig. 1).

(*ii*) The corresponding digit was in error in the generator set. The probability of this event is bounded by p_1.

(*iii*) An error was made in determining the operation to be performed. This probability is denoted by p_{control}.

(*iv*) An error was made in performing the desired operation. This probability is denoted by $p_{\text{operation}}$.

(*v*) An error was made in performing the AND operation. This probability is denoted by p_{AND}.

(*vi*) An error occurred within the memory in the result set. The probability of this event is bounded by $2p_r$.

Therefore, by the union bound

$$\Pr[\text{error in any digit within the registers in the result set}] < 3p_1 + p_{\text{control}} + p_{\text{operation}} + p_{\text{AND}} + 2p_r$$

and we require that this sum of probabilities be less than p_0. In order to show that it is possible to satisfy this inequality, let us consider a numerical example where

$$p_{\text{control}} = p_{\text{operation}} = p_{\text{AND}} = p_a = p_r = p_d = 10^{-10},$$
$$p_0 = 10^{-8},\ J = 14 \text{ and } K = 15.$$

For this example $p_1 \approx 2 \times 10^{-10}$, therefore

$$3p_1 + p_{\text{control}} + p_{\text{operation}} + p_{\text{AND}} + 2p_r \approx 1.1 \times 10^{-9} < 10^{-8} = p_0$$

showing that the condition on p_0 is satisfied.

The probability that a memory failure or a propagation failure occurs in any one of the memories in the result set can be computed by exactly the same method used in the previous section to compute the probability of a memory failure in the stable result memory following the operation of vector addition modulo-2. In fact the result is exactly the same as that obtained in the previous section since introducing the bound p_0 makes all the relevant probabilities iden-

tical. The result is as follows

Pr[memory failure or propagation failure in one memory in result set]

$$< \frac{C}{[1 - J/K]^2} \cdot k^{-\beta+3}.$$

Therefore, the probability that any memory failure or propagation failure occurs anywhere in the result set is bounded by

Pr[memory failure or propagation failure in result set]

$$< \frac{C}{[1 - J/K]^2} \cdot k^{-\beta+4}.$$

The final step consists of performing the operation of vector addition modulo-2 on the contents of the memories in the result set. The adder network, shown in Fig. 4, performs $k-1$ vector additions modulo-2. According to the results of the previous section, the probability that any one of these operations is performed incorrectly is bounded by

Pr[error in any of $k - 1 \oplus$ operations]

$$< (k - 1) \frac{C}{[1 - J/K]^2} \cdot k^{-\beta+3}$$

$$< \frac{C}{[1 - J/K]^2} \cdot k^{-\beta+4}.$$

The result is in error only if a memory failure or propagation failure occurs during the first m iteration, during the computation, or during the vector additions modulo-2. Therefore, the probability that the result is in error is bounded by

Pr[result is in error]

$$< 6\left\{\left[\frac{J}{1 - J/K}\right] + 1\right\} \cdot \left\{\frac{C}{1 - J/K} \cdot \log \left[\frac{k}{1 - J/K}\right]\right\} k^{-\beta+3} + \frac{2C}{[1 - J/K]^2} \cdot k^{-\beta+4} \qquad \text{for} \quad k > 2$$

$$\triangleq P_{\text{result}}.$$

Finally, we must bound the probability that an error occurs during a sequence of 3 operations. This probability is bounded by the sum of the probabilities that an error occurs on any one of the 3 operations. Therefore,

Pr[error during a sequence of $\mathfrak{z}$ operations] $< \mathfrak{z} \cdot P_{result}$.

The dominant term in this bound is proportional to $k^{-\beta+4}$ where β is a constant for any particular sequence of computing systems. Ref. 1 gives a numerical example where β equals 7.55. For this example, the probability that an error occurs in a sequence of $\mathfrak{z}$ operations is bounded by a function which decreases as k^{-3}. Therefore, for the sequence of computing systems for which $\beta = 7.55$, the probability of an error during $\mathfrak{z}$ operations can be made arbitrarily small by making k sufficiently large.

Thus far we have not considered the complexity of these computing systems. The "basic processor," that is, the machine that performs one operation on each of the k pairs of digits has a complexity which is proportional to k^2. Let us suppose that the system S_k is capable of performing a sequence of $\mathfrak{z}$ operations on each of the k digits. In general, this requires $\mathfrak{z}$ basic processors. Thus the complexity of the system is proportional to $\mathfrak{z} \cdot k^2$. The amount of computation, the number of operations performed on pairs of digits, is equal to $\mathfrak{z} \cdot k$. Thus, for S_k, the ratio of the amount of computation to the complexity, is proportional to k^{-1}. The computing capacity equals the maximum value of this ratio for which the probability of error can be made arbitrarily small. Since the probability of error for S_k is proportional to $\mathfrak{z} \cdot k^{-\beta+4}$, this probability of error can be made arbitrarily small only in the limit as k approaches infinity; but in this limit the ratio described above approaches zero. Thus the computing capacity for these systems equals zero.

V. ARITHMETIC OPERATIONS ON OPERANDS OF BOUNDED MAGNITUDE

The systems described in the previous section have two major shortcomings. The first is that the computing capacity equals zero and the second is that the systems are restricted to performing operations on corresponding bits in the operands. Let us consider the first of these shortcomings. We would like to show that it is possible to modify these systems in such a way that the amount of computation per component is independent of k, whereas the probability of error decreases with increasing k. In order to obtain such a result, we must reuse the basic processor. This means that we must be able to "program" the computing system. We have already shown that it is possible to "program" the basic processor to perform different operations. We must now show that it is possible to store the program and the operands in memories which can be located when the contents are needed.

In the previous section we described a method for locating desired

generators which were stored within the generator set. The jth generator was found by first setting up an "address set" consisting of $J \cdot N$ stable memories each of which has a one in the jth information symbol position and zeros in all other information symbol positions. After all stable memories had performed m iterations, the generators were ANDed with the address set and the result was propagated through a modulo-2 addition network as shown in Fig. 4. We can store the program and operands in a "program set" consisting of k memories and use precisely the same method to locate a memory in this set. Thus, the "address" of the jth memory in the program set is a 1 in the jth information position of a code word.

In order to keep track of the next operation to be performed, we need to use one memory as an "instruction counter." Initially this memory contains a code word with a 1 in the first information position and a zero in all other information positions. After each operation, we shift the information digits one position to the right. A simple modification of the basic processor allows it to perform this shift operation. In order to specify whether a shift is desired we add one more operation-code set. The information digits in the memories within this additional set are all 1's if a shift is desired and all 0's otherwise. (Notice that we are really wasting $(k-1)$ of the information digits in each memory in this additional set since only one digit is required to specify whether a shift is desired).

The processor checks the "shift bit" before performing any operation to see if a shift is desired. If the shift bit equals 1, the shift is performed on the first operand and no other operations are performed. If the shift bit is 0, the operations specified by the other operation code sets are performed. In either case, the result is computed as before by adding the appropriate generators.

The second major shortcoming of the systems described in the previous section is that only operations on corresponding bits in the operands can be performed. In order to perform more general operations we might consider permuting the digits in one of the operands before the operation is performed. Unfortunately, the probability that a particular digit is permuted incorrectly depends on k and, in fact, approaches 1/2 as k approaches infinity. This problem arises because we have attempted to perform one operation that involves all k information digits, but it can be avoided by dividing the k digits into a number of segments where the number of digits in each segment does not depend on k; that is, the number of segments must grow with k. We can then treat each segment as a separate operand. This

means that we must restrict our attention to operations performed on operands of bounded magnitude, which is certainly not a severe restriction.

Let us consider the digits in one particular segment. If it were possible to permute these digits before each operation, we could compute the sum or product of the digits in this segment or, in fact, any finite sequence of arithmetic operations on these digits by performing a sequence of bit-by-bit operations on the appropriately permuted digits. Consider the additional modifications that must be made to the basic processor in order to allow it to perform these permutations. For the purpose of specifying the desired permutation we must again increase the number of operation-code sets. If the longest segment contains s digits, the permutation can be described by the contents of the memories in $\log s$ additional operation-code sets. The permutation of a particular segment of digits is specified by the corresponding segments of the memories in these additional operation-code sets. (Notice that a permutation of s digits can be described by $s \cdot \log s$ digits.) The operation is performed according to the method described previously. However, in this case the information bits within each segment of one operand are permuted just before the actual operation is performed.

Let us consider how these modifications affect the bound on the probability of error for the basic processor. This bound is given in Section IV. The first term bounds the probability that either a memory failure or a propagation failure occurs anywhere within the operand sets, the operation code sets, or the generator set during the first m iterations. Since there are now $[\log s] + 1$ additional operation-code sets, the coefficient of this first term must be changed from

$$\left[6\left(\frac{J}{1 - J/K}\right) + 1\right] \quad \text{to} \quad \left[(7 + \log s)\left(\frac{J}{1 - J/K}\right) + 1\right].$$

However, the dependence on k of the first term is unchanged.

The second term bounds the probability that a memory failure occurs either in the result set or in one of the memories required for the modulo-2 addition operation. In deriving this second term, it was necessary to bound the probability of error for digits in the result set. We considered a set of events at least one of which must have occurred if a particular digit is in error. There are now two additional events which could lead to such an error. The first is that the shift instruction was interpreted incorrectly and the second is that the permutation operation was performed incorrectly. In both

cases, the probability of error per digit is independent of k. Therefore, it is possible to find examples where p_0 bounds the probability of error for digits in the result set.

For example, consider the numerical values presented in Section IV. If the probability of a permutation error equals 10^{-10} and the probability of a shift error also equals 10^{-10}, then $p_0 = 10^{-8}$ is still a bound on the probability of error per digit. Thus the second term is not changed by the system modification. Therefore the dominant term in this probability of error is still proportional to $k^{-\beta+4}$. The equipment required to select the appropriate memory from the program set is similar to a basic processor. Therefore, the probability of making an error in one selection operation is of the same form as the probability of error for the basic processor. In particular the dominant term in this probability of a selection error is proportional to $k^{-\beta+4}$.

Finally, let us consider the number of operations that can be performed by this modified computing system. We have allowed k memories for storing the program and the operands. This means that the number of steps in the computation can be at least proportional to k where each step consists of one set of k operations performed by the basic processor. Therefore the amount of computation can be proportional to k^2. Furthermore, the complexity of the system is also proportional to k^2 since only one basic processor and one operation selector is needed. Therefore, the amount of computation per component is independent of k and in general is nonzero.

The probabibity of an error during the k steps in the computation is upper bounded by k times the probability that an error occurs during any one step. Therefore, this overall probability of error is proportional to $k^{-\beta+5}$. Since we have already presented an example where $\beta = 7.55$, in this case the probability of error can be made arbitrarily small by making k sufficiently large while keeping the redundancy fixed. This shows that the computing capacity for systems of this type is nonzero.

VI. CONCLUSIONS

There are basic processors, designed entirely from unreliable components, which can perform arbitrary binary operations on the corresponding information bit of two operands stored in stable memories. These information bits can be shifted, by one bit, or permuted, within segments of bounded length, before performing the operation. The probability of error for the basic processor can be made to vanish

as $k^{-\beta+4}$ where k is the number of information bits in each operand and β is a function of the component malfunction probabilities and the information rate of the code used in storing information in the memories.

A program, including the necessary operands, can be stored in k stable memories and the operations specified by the program can be performed in sequence by a single basic processor. It is possible to perform arbitrary k-step computations on numbers of bounded magnitude. Furthermore, the number of these computations that can be performed simultaneously is proportional to k. The complexity of the equipment required to perform the computations and the amount of computation are both proportional to k^2. Thus the amount of computation per component and hence the computing capacity is nonzero.

These results apply to systems designed from either of two types of unreliable components. The first type is one which malfunctions because it is perturbed by random noise in the system. The malfunctions of these components are modeled mathematically by assuming that they are statistically independent from one component to another and from one use of a particular component to another use. The second type of component is one which fails permanently; however, it is assumed that components which have failed are regularly replaced by good ones. In this case the mathematical model is based on the assumptions that components fail independently of each other and that there is an upper bound, of value less than $\frac{1}{2}$, on the probability of malfunction of any particular component on any single use.

All the results are existence proofs. Therefore, the criterion for choosing the particular systems to be considered was simplicity of analysis rather than efficiency of operation. Furthermore, the emphasis was not placed on obtaining the tightest possible bounds but rather on obtaining simple bounds that were sufficient to prove the desired results. In particular, one would hope that the probability of error would decrease exponentially with k rather than algebraically with k. However, the particular techniques used here do not lead to such a bound.

There are many questions still to be considered concerning the design of practical systems constructed from unreliable components. There are also many theoretical questions concerning the derivation of bounds which are tighter than those derived here. It is hoped that the development of practical, reliable computing systems will follow the presentation of existence theorems like those presented here just

as the development of practical, reliable communication systems followed the presentation of the existence theorems of information theory.

VII. ACKNOWLEDGMENTS

I would like to sincerely thank Professor Robert M. Fano, who supervised this work, for suggesting the approach to the problem and supplying guidance and encouragement throughout the research. I also wish to thank Professor Robert G. Gallager for his help. Many of the results presented here are based on results originally obtained by him. Finally, I wish to thank Professors Peter Elias and Claude E. Shannon who contributed valuable suggestions and constructive criticism.

APPENDIX

A Bound on the Probability of Error

In this appendix we upper bound the probability of error for one vector addition modulo-2. The method for deriving this bound is based on the one used in Ref. 1 to upper bound the probability of a memory failure. Since the method is long and rather involved, it is not reviewed here. Please see the proof in Ref. 1, because here we discuss only those places where the two proofs differ. Both the terminology and the notation here is the same as in Ref. 1.

The computing systems being considered are described in Section III. It is assumed that the operation was performed at $t = T$. The operation takes t_{op} seconds to be performed. Some time during the τ seconds following $t = T + t_{op}$, the stable result memory starts to perform the first correcting cycle on the newly inserted digits. We denote the time at which this correcting cycle starts by $t = T + \sigma$ (see Fig. 5).

It is assumed that neither a memory failure nor a propagation failure has occurred in either operand memory up to $t = T$. A bound has already been derived on the probability of error per digit in the result memory. We must now bound the probability of a propagation failure occurring in the result memory. This involves relating the Δ configurations for the result memory to the Δ configurations for the two operand memories. Finally, the probability of a propagation failure will be used to obtain the desired bound on the probability of making an error in performing the operation.

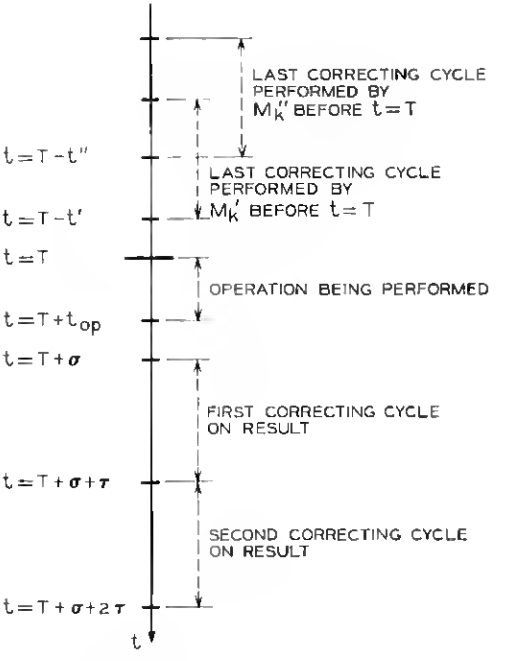

Fig. 5 — Sequence of events pertaining to $\oplus$ operation.

To relate the Δ configurations for the result memory to those for the two operand memories, let us consider the type $i (i > 0)$ Δ configuration for the result memory evaluated at $t = T + \sigma$. If there is a 1 in the entry corresponding to the digit d_0, the controlling errors must have changed the value of d_0 in the result memory; but this change can occur only if the controlling errors changed the value of one but not both digits in position d_0 in M_k' and M_k''. If both these digits in position d_0 had been changed, the two changes would cancel each other when the operation of vector addition modulo-2 was performed. Therefore, the type $i (i > 0)$ Δ configuration for the result memory evaluated at $t = T + \sigma$ is equal to the vector sum modulo-2 of the type i Δ configurations for M_k' and M_k'' where the Δ configuration for M_k' (or M_k'') is evaluated at the end of the last correcting cycle performed before $t = T$ (that is, at $t = T - t'$ or $t = T - t''$).

The type 0 Δ configuration evaluated at $t = T + \sigma$ is computed in a different way. A 1 in this type 0 Δ configuration represents a new error in the stable result memory at $t = T + \sigma$. To compute the configuration of new errors, we imagine that the computing system was

noiseless up to the beginning of the most recent correcting cycle performed on the digits stored in the result memory at $t = T + \sigma$. In this case, the "most recent correcting cycle" consists of the last correcting cycle performed by M_k' before $t = T$ and the last correcting cycle performed hy M_k'' before $t = T$ (see Fig. 5).

Since the probability of error per digit at $t = T + \sigma$ can be hounded by p_0, the probability of a new error at $t = T + \sigma$ can also be bounded by p_0. Tberefore the probability of a 1 in the type 0 Δ configuration evaluated at $t = T + \sigma$ must be less than p_0. Since the type $i (i > 0)$ Δ configuration for the result memory evaluated at $t = T + \sigma$ is the modulo-2 sum of the type i Δ configurations for M_k' and M_k'', the probability of a 1 in the type i Δ configuration evaluated at $t = T + \sigma$ is hounded hy twice the probability of a 1 in the type i Δ configuration for M_k' (or M_k'') evaluated at $t = T - t'$ (or $t = T - t''$).

The method for computing a bound on the probability of a memory failure at $t = T + \sigma$ is exactly the same as that used to compute a bound on tbe probability of a memory failure at any other time for any stable memory. This method consists of bounding the probability that a noiseless correcting network could correct all errors present at $t = T + \sigma$ by performing m iterations. This probahility is bounded by the probability that the initial propagation failure occurs at $t = T + \sigma$ or during the m noiseless iterations performed after $t = T + \sigma$.

The initial propagation failure occurs whenever there are one or more 1's in the type m Δ configuration. Since, by assumption, no propagation failure occurred in either M_k' or M_k'' before $t = T$, the type m Δ configuration for M_k' and M_k'' must be all zero for all $t < T$. The type m Δ configuration for the result memory evaluated at $t = T + \sigma$ is the vector sum modulo-2 of two of these type m Δ configurations for M_k' and M_k''. Therefore, the type m Δ configuration evaluated at $t = T + \sigma$ must be all zero. Hence, no propagation failure can occur at $t = T + \sigma$.

A hound is derived in Ref. 1 on the probability of a 1 in the entry corresponding to the digit d_0 in the type m Δ configuration evaluated at $t = L\tau$. The equation used to compute this bound is;

$$\Pr[1 \text{ in one particular entry in type } m \ \Delta \text{ configuration evaluated after } L\text{th iteration}]$$

$$< (J-1)(K-1)\binom{J-2}{J/2-1}[(K-1)(2p_0)]^{J/2-1} \cdot \Pr[1 \text{ in one particular entry in type } m-1 \ \Delta \text{ configuration evaluated after } (L-1)\text{th iteration}] \quad (1)$$

where p_0 is both a bound on the probability of error for digits in the registers of M_k and a bound on the probability of a 1 in the type 0 Δ configuration. To compute a bound on the probability that the initial propagation failure occurs at $t = T + \sigma + \tau$, we apply this recurrence relation. By applying it $m - 1$ times, we obtain the probability of a 1 in the type $(m - 1)$ Δ configuration for M_k' evaluated at $t = T - t'$, namely

$$\begin{aligned}\Pr[&1 \text{ in one particular entry in type } (m-1)\ \Delta \text{ configuration evaluated at } t = T - t'] \\ &< p_0\left\{(J-1)(K-1)\binom{J-2}{J/2-1}[(K-1)(2p_0)]^{J/2-1}\right\}^{m-1} \\ &\triangleq P_{m-1}\ .\end{aligned}$$

The probability of a 1 in the type $(m - 1)$ Δ configuration evaluated at $t = T + \sigma$ is bounded by $2 \cdot P_{m-1}$. One final application of tbe recurrence relation leads to

$$\begin{aligned}\Pr[&1 \text{ in one particular entry in type } m\ \Delta \text{ configuration evaluated at } t = T + \sigma + \tau] \\ &< 2p_0\left\{(J-1)(K-1)\binom{J-2}{J/2-1}[(K-1)(2p_0)]^{J/2-1}\right\}^{m}.\end{aligned}$$

Gallager has shown that there are low-density parity-check codes[2] for which

$$\frac{\log\left[\frac{k}{2K} - \frac{k}{2J(K-1)}\right]}{2\log(J-1)(K-1)} < m < \frac{\log\left[\frac{k}{1-J/K}\right]}{\log(J-1)(K-1)}.$$

Therefore,

$$\begin{aligned}\Pr[&1 \text{ in one particular entry in type } m\ \Delta \text{ configuration evaluated at } t = T + \sigma + \tau] \\ &< 2p_0\left\{(J-1)(K-1)\binom{J-2}{J/2-1}\right. \\ &\qquad \left.\cdot[(K-1)(2p_0)]^{J/2-1}\right\}^{\log\left[(k/2K)-[k/2J(K-1)]\right]/2\log(J-1)(K-1)} \\ &= 2p_0\left\{\left[\frac{1}{2K} - \frac{1}{2J(K-1)}\right]k\right\}^{-\beta}\end{aligned}$$

where

$$\beta \triangleq \frac{\log\left\{(J-1)(K-1)\binom{J-2}{J/2-1}[(K-1)(2p_0)]^{J/2-1}\right\}}{2\log(J-1)(K-1)}.$$

The probability that the initial propagation failure occurs at $t = T + \sigma + \tau$ is bounded by the probability of one or more 1's in the type m Δ configuration evaluated at $t = T + \sigma + \tau$. Since there are $J \cdot N$ entries in this Δ configuration, and since $N \leq k/(1 - J/K)$, by the union bound,

$$\begin{aligned}\Pr[\text{initial propagation failure occurs at } t = T + \sigma + \tau] &< J \cdot \left[\frac{k}{1 - J/K}\right] \cdot 2p_0 \left\{\left[\frac{1}{2K} - \frac{1}{2J(K-1)}\right]k\right\}^{-\beta} \\ &\triangleq 2Ck^{-\beta+1}.\end{aligned}$$

This same bound applies to the probability that the initial propagation failure occurs on any one of the m iterations performed after $t = T + \sigma$.

A memory failure can occur at $t = T + \sigma$ only if the initial propagation failure occurs on one of the m iterations performed after $t = T + \sigma$. Therefore, by the union bound,

$$\begin{aligned}&\Pr[\text{memory failure at } t = T + \sigma \mid \text{no memory failure or propagation failure for } t < T] \\ &= \Pr[\oplus \text{ operation is performed incorrectly}] \\ &< 2 \cdot C \cdot m \cdot k^{-\beta+1} \\ &< 2 \cdot C \cdot \log\left[\frac{k}{1 - J/K}\right] \cdot k^{-\beta+1} \\ &< 2 \cdot \frac{C}{1 - J/K} \cdot k^{-\beta+2}.\end{aligned}$$

To show that the computing systems being considered are reliable, we must show that it is possible to perform a sequence of 3 operations with an overall probability of error that can be made arbitrarily small by choosing k sufficiently large. After one iteration of the stable result memory (that is for $t \geq T + \sigma + \tau$), the result stored in this memory can be used as an operand for another vector addition modulo-2. Let us suppose that another addition operation is performed at $t = T + \sigma + \tau \triangleq T'$ and that the first correcting cycle on this second result starts at

$t = T' + \sigma'$. We now bound the probability that this second operation is performed incorrectly given that the first operation was performed correctly (that is, given that no memory failure or propagation failure occurred at $t = T + \sigma$). The method for deriving this bound is identical to the one used above. We bound the probability that the initial propagation failure occurs at $t = T' + \sigma'$ or on any one of the next m iterations. In this case, the probability that the initial propagation failure occurs at $t = T' + \sigma'$ is not zero since a propagation failure could have occurred at $t = T'$.

Equation 1 relates the probability of a 1 in the type i Δ configuration evaluated after one particular correcting cycle, to the probability of a 1 in the type $(i - 1)$ Δ configuration evaluated after the previous correcting cycle. The application of this equation is simple except in cases where an addition operation has been performed between successive correcting cycles. It has been shown already that in each case where an addition operation has been performed, we must double the probabilities that otherwise would have been substituted into the equation. Since, in this case, there were two addition operations performed within the m correcting cycles of interest, the value of this bound must be twice the value of the bound derived above.

Therefore,

$$\begin{aligned}\Pr[\text{second } \oplus \text{ operation is performed incorrectly} \mid \text{first } \oplus \\ \text{operation was performed correctly}]\end{aligned}$$

$$< 2^2 \cdot [m + 1] \cdot C \cdot k^{-\beta+1}$$

$$< 2^2 \cdot C \cdot \log \left[\frac{k}{1 - J/K}\right] \cdot k^{-\beta+1}$$

$$< 2^2 \cdot \frac{C}{1 - J/K} \cdot k^{-\beta+2}$$

where we have used the bound

$$m + 1 < \frac{\log \left[\dfrac{k}{1 - J/K}\right]}{\log (J - 1)(K - 1)} + 1 < \log \left[\frac{k}{1 - J/K}\right] \quad \text{for} \quad k > 2.$$

This result can be extended to any number of vector additions modulo-2 performed in series. However, since only m iterations are considered in deriving this bound, the largest value that this bound can have corresponds to the case where an addition operation has

been performed between each iteration for the last m iterations. For this "worst case" the bound on the probability that the mth vector addition modulo-2 is performed incorrectly, given that all previous additions were performed correctly, is

$$\begin{aligned}\Pr[\text{any } \oplus \text{ operation is performed incorrectly} \mid \text{all previous } \oplus \text{ operations were performed correctly}] \\ < 2^m \cdot C \cdot \log\left[\frac{k}{1 - J/K}\right] \cdot k^{-\beta+1} \\ < \frac{C}{1 - J/K} \cdot \log\left[\frac{k}{1 - J/K}\right] \cdot k^{-\beta+2} \\ < \frac{C}{[1 - J/K]^2} \cdot k^{-\beta+3}.\end{aligned}$$

In this Appendix we have not made any reference to the underlying assumptions concerning the models for component malfunctions. We are interested in two models as explained in the introduction. Since equation 1 applies for either of these models and since the mathematical development based on equation 1 also applies for either model, the final result, namely the probability of the initial modulo-2 operation error, has the form given above for either component malfunction model.

REFERENCES

1. Taylor, M. G., "Reliable Information Storage in Memories Designed from Unreliable Components," B.S.T.J., *47*, No. 10 (December 1968), pp. 2299–2337.
2. Gallager, R. G., *Low-Density Parity-Check Codes*, Cambridge, Massachusetts: MIT Press, 1963.
3. Peterson, W. W., *Error Correcting Codes*, New York: The MIT Press and John Wiley and Sons, Inc., 1961.